performed to determine if there is a match flag associated with the complete icon.

The following figures explain the "icon algebra" used in implementing the invention. FIG. 7 is a flow chart of the steps used in an icon shift function in accordance with one embodiment of the invention. The shift module determines the transform for a shifted message (i.e., "A0" or $X^zA(x)$). Where X^z means the function is shifted by z places (zeros) and A(x) is a polynomial function. The process starts, step 120, by receiving the transform 122 to be shifted at step 124. Next the a pointer 126 is extracted at step 128. The transform 122 is then moved right by the number of bits in the pointer 126, at step 130. This forms a moved transform 132. Note the words right and left are used for convenience and are based on the convention that the most significant bits are placed on the left. When a different convention is used, it is necessary to change the words right and left to fit the convention. Next the moved transform 132 is combined (i.e., XOR'ed) with a member 134 associated with the pointer 126, at step 136. The member associated with the pointer is found in a transform look table, like the one shown in FIG. 11. Note that this particular lookup table is for a CRC-32 polynomial code, however other polynomial codes can be used and they would have different lookup tables. This forms the shifted transform 138 at step 140, which ends the process at step 142. Note that if the reason for shifting a first transform is to generate a first-second transform then first transform must be shifted by the number of bits in a second data string. This is done by executing the shift module X times, where X is equal to the number of data bits in the second data

string divided by the number of bits in the pointer. Note that another way to implement the shift module is to use a polynomial generator. The first transform 122 is placed in the intermediate remainder register. Next a number of logical zeros (nulls) equal to the number of data bits in second data string are processed.

FIG. 8 is a flow chart of the steps used in an icon unshift function in accordance with one embodiment of the invention. An example of when this module is used is when the transform for the data string "AB" is combined with the transform for the data string "B". This leaves the transform for the data string "A0" or $X^zA(x)$. It is necessary to "unshift" the transform to find the transform for the data string "A". The process starts, step 150, by receiving the shifted transform 152, at step 154. At step 156 a reverse pointer 158 is extracted. The reverse pointer 158 is equal to the most significant portion 160 of the shifted transform 152. The reverse pointer 158 is associated with a pointer 162 in the reverse look up table (e.g., see FIG. 12) at step 164. Next, the member 166 associated with the pointer 162 in the table of FIG. 11 for example, is combined with the shifted transform at step 168. This produces an intermediate product 170, at step 172. At step 174 the intermediate product 170 is moved left to form a moved intermediate product 176. The moved intermediate product 176 is then combined with the pointer 162, at step 178, to form the transform 180, which ends the process, step 182. Note that if the number of bits in the "B" data string (z) is not equal to the number of bits in the pointer then the unshift module is executed X times, where X=z/(number of bits in pointer).

09572565 . 040201

FIG. 9 is a flow chart of the steps used in a transform function in accordance with one embodiment of the invention. The transform module can determine the first-second transform for a first-second data string given the first transform and the second data string, without first converting the second data string to a second transform. The process starts, step 190, by extracting a least significant portion 192 of the first transform 194 at step 194. This is combined with the second data string 196 to form a pointer 198, at step 200. Next a moved first transform 202 is combined with a member 204 associated with the pointer in the look up table (e.g., FIG. 11), at step 206. A combined transform 208 is created at step 210 which ends the process, step 212. Note that if the pointer is one byte long then the transform module can only process one byte of data at a time. When the second data string is longer than one byte then the transform module is executed one data byte at a time until all the second data string has been executed. In another example assume that first transform is equal to all zeros (nulls), then the combined transform is just the transform for the second data string. In another embodiment the first transform could be a precondition and the resulting transform would be a precondition-second transform. In another example, assume a fourth transform for a fourth data string is desired. A first data portion (e.g., byte) of the fourth data string is extracted. This points to a member in the look up table. When the fourth data string contains more than the first data portion, the next data portion is extracted. The next data portion is combined with the least significant portion of the member to form a pointer. The member is then moved right by the number of bits in the next data portion to

09572596 .101201